

Programming Plantation Lines on Driverless Tractors

Antonio Elias Fabris¹
Marcelo Zanchetta do Nascimento²
Valério Ramos Batista³

¹IME-USP, r. Matão 1010, 05508-090 São Paulo-SP, Brazil; aef@ime.usp.br

<https://sites.google.com/site/aefabris>

²FACOM-UFU, av. João Naves de Ávila 2121, Bl.A, 38400-902 Uberlândia-MG, Brazil; nascimento@facom.ufu.br

<http://www.facom.ufu.br/~nascimento>

³CMCC-UFABC, r. Sta. Adélia 166, Bl.B, 09210-170 St. André-SP, Brazil; vramos1970@gmail.com

<https://sites.google.com/site/vramos1970>

Abstract

Recent advances in Agricultural Engineering include image processing, robotics and geographic information systems (GIS). Some tasks are still accomplished manually, like drawing plantation lines that optimize productivity. Herewith we present an algorithm to find the optimal plantation lines in linear time. The algorithm is based upon classical results of Geometry which enabled a source code with only 573 lines. We have implemented it in Matlab for sugar cane, and it can be easily adapted to other crops like coffee, maize and soy.

1 Introduction

The spread of technological resources have been improving almost all human activities: Education, Medicine, Architecture, etc. Regarding the sectors of the Economy, they are all increasingly dependent on cutting-edge technology. From top to bottom, they run from the Quinary Sector (top executives in government, healthcare, media) to the Primary Sector (mining, farming, fishing, agriculture).

Any economic activity is classified in one of these sectors. For instance, Engineering belongs to the Secondary Sector. By the way, we would like to comment on Electronic Engineering for two reasons: it is the source of Technology itself, and also an example in which Technology is self-applied. For instance, electric circuits designed through Very-large-scale integration (VLSI) are composed of *millions* of terminals. It is impracticable to design and test these circuit projects without the help of specific software. See [3] for an example.

In the case of Agriculture there are innumerable examples in which modern technology is applied. For instance, robotics [12], image processing [2, 6] and GIS [9, 1].

However, some tasks are still accomplished manually. This is the case of finding the ideal plantation lines for certain crops like sugar cane, maize, coffee and soy. Sugar cane is particularly important for two reasons. First because its annual production is concentrated by three leading countries: Brazil, India and China. In million of tonnes they yield ca. 700, 300 and 100, respectively. Second because this crop is an important source of ethanol fuel as a substitute to petrol.

1.1 Ideal Plantation Lines

Herewith we present an algorithm to draw ideal plantation lines for sugar cane. It can be adapted to other crops, but its implementation was specially devoted to programming the plantation lines on driverless tractors that groove, plant and harvest the sugar cane. In order to optimize productivity, the following conditions must be satisfied:

1. the tractor cannot slant up or down more than 5° while driving;
2. the tractor cannot make curves that are under $50m$ in radius;
3. two neighbouring grooves must be $3m$ distant from each other;
4. there must be the least possible number of plantation lines;
5. on a sloping land the extremities of each plantation line must lie above some of its interior stretches.

Such conditions are based on machine and agricultural restrictions. Conditions (1) and (2) prevent the tractor from turning over, whereas (3) and (4) yield the maximum amount of sugar cane within the plot that minimize tractor manoeuvres from one plantation line to the other. These manoeuvres are performed at the boundary of the plot. Finally, condition (5) guarantees that rain water will flow to the inside of the plot. Namely, on a sloping land plantation lines should *never* coincide with level curves. Otherwise the water will puddle in the grooves.

510662.36	7920042.53	100.00
510658.26	7920032.13	100.00
510700.36	7920086.57	99.00
510689.00	7920058.88	99.00
510681.14	7920035.64	99.00
510668.97	7920004.84	99.00
510664.13	7919995.51	99.00
510662.18	7919992.77	99.00
510722.89	7920088.81	98.00
510707.74	7920051.87	98.00

1.2 Main Contributions of this Paper

To the best of our knowledge, an algorithm that solves the proposed problem was never published before. One of the reasons come from conditions (2) and (3). Since plantation lines are parallel, we can begin with some of them that satisfy (3) but further parallels risk having osculating circles with radii lesser than $50m$. So back-and-forth tests are required and they normally result in a tedious manual process of trial-and-error. We reduce the problem to a robust algorithm which has linear time complexity.

Another reason is that in practice about 50% of the plots are *concave* regions. On the one hand, we have obtained a programme that works automatically for convex plots, and the users intervene only for concave plots, in which case they have to decide very little. This characterizes a semi-supervised algorithm. On the other hand, we believe that an unsupervised algorithm, if it ever can be found, would unnecessarily increase the complexity of the programme. The difficulty in developing such a method would make the ratio cost/benefit considerably high.

Finally, although condition (3) implies that the manual drawing of a single plantation line is enough to determine all of them, the optimization problem cannot be solved automatically unless we equate the terrain. Its topography is given as a datafile of coordinates, which are graphically rendered by commercial software of Computer-Aided Design (CAD). Our algorithm includes a quick 2D-interpolation of these coordinates and we work on a terrain equated by the graph of a polynomial $z(x, y)$. By the way, it is worthwhile to mention that most CAD/CAM commercial softwares devoted to agriculture can easily integrate our programme.

2 Preparing the Input Data

2.1 Obtaining the Topography

Typically, GIS can be used to locate points in a region. However, in the case of plantations we need more accurate data. Hence GIS is useful for fixing a standard meridian and parallel of the Earth, and they will be referential for the x and y coordinates, respectively.

Precise (x, y, z) coordinates of the terrain are then obtained by an unmanned aerial vehicle (UAV). They are stored in a datafile named `level_curves<n>.txt`, where n labels a specific terrain. Our programme is compressed in the file `tml.zip` that also contains 5 different examples of terrains. Some consecutive lines of `level_curves3.txt` are shown in Table 2.1.

From Table 2.1 we see that level curves are ordered by the representing height.

The plot boundary is in fact a polygon, of which the (x, y) -vertices are stored in `plot<n>.txt` (n must refer to the same previous number). Our file `tml.zip` also contains the corresponding 5 different plots. A plot-file is like a level curves-file, except for having far fewer points and only two columns (the representing height is omitted).

2.2 Equating the Terrain

There are many well-known algorithms devoted to 2D-polynomial interpolation. In Matlab the function `interp2` makes use of some methods as the C^2 cubic spline (see [4]) and the C^1 cubic convolution (see [8]). They result in smooth 2D-functions that are piecewise defined by 2D-polynomials.

Usually the input level curves-files obtained by the UAV contain points in hundreds for an accurate numerical description of the terrain. However, we just need a sample of these points because a plantation area has to be quite regular. Indeed, in practice any plot can be confined to a square of edgelenh $700m$. By calling the highest and the lowest level of a plot H and L respectively, then we must have $0 \leq H - L \leq 40m$.

Moreover, the plot ought not to be hilly and therefore both ways along the plot boundary from H to L should be monotonically decreasing. This last condition is however just recommendable. We shall see some plots that violate it.

Anyway, the regularity of a plot extends to the terrain. This is because the UAV flies over a rectangular area that is just big enough to contain the plot. Therefore, the (x, y) -coordinates within any level curves-file fit in a rectangle. See our 2D-rendering of `level_curves1.txt` in Figure 1.

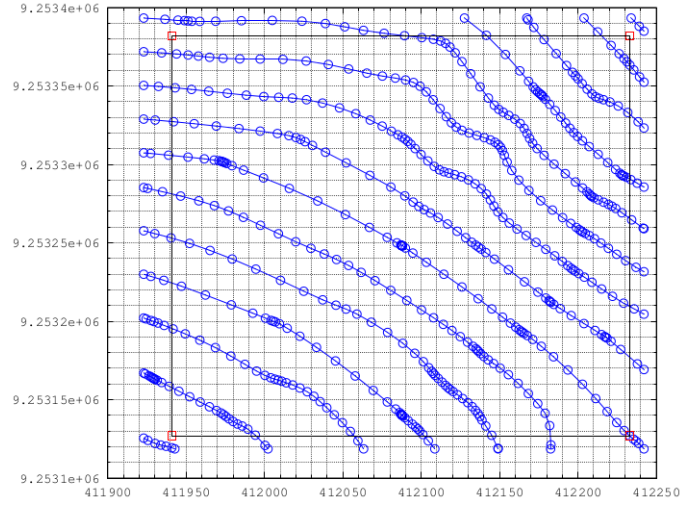


Figure 1: 2D-rendering of a level curves-file.

In Figure 1 each tiny circle is centred at a certain (x, y) of the data file. Connected circles have the same z -coordinate. This example has a very simple plot, which is the inner rectangle from $x = 411,940.91$ to $x = 412,232.99$ and from $y = 9.25313e6$ to $y = 9.25338e6$. Vertices of a plot are always marked with tiny squares.

As explained before, we just need to interpolate a sample of the (x, y, z) -points. In our programme the extreme coordinates of the terrain are called X_{max} , X_{min} , Y_{max} and Y_{min} . In practice, we get a 2D-polynomial out of a subgrid of at most $5 \cdot 5 = 25$ points equally spaced in $[X_{min}, X_{max}] \times [Y_{min}, Y_{max}]$.

Hence the subgrid is an $N \times M$ matrix, and for each point $(x(i, j), y(i, j))$ of the subgrid there is a closest pair (x, y) in the level curves-file, where $1 \leq i \leq N \leq 5$ and $1 \leq j \leq M \leq 5$. We take its corresponding z to define $Z(i, j)$. Finally, we get the terms of our polynomial $z(x, y)$ as $q(i, j)x^{M-j}y^{N-i}$, where the coefficients $q(i, j)$ are obtained in Algorithm 1.

ALGORITHM 1: Obtaining the coefficients $q(i, j)$ of the polynomial $z(x, y)$

Input: M, N, x, y, Z

Output: q

for $i=1:N$ **do**

$p(i,:) = \text{polyfit}(x(i,:), Z(i,:), M-1);$

end

for $j=1:M$ **do**

$q(:,j) = \text{polyfit}(y(:,j), p(:,j), N-1);$

end

Notice that we use the Matlab function `polyfit` twice in order to get a single 2D-polynomial $z(x, y)$. Of course, `polyfit` is devoted to 1D-interpolation. However, its double application is feasible because we have at most 25 points.

Figures 2 and 3 were obtained from `level_curves1.txt`, and they exemplify how well the input data are reproduced by Algorithm 1.

3 The Master Line

As remarked in the Introduction by condition (3), which is due both to agricultural and machine reasons, plantation lines should be parallel. Thus a single plantation line determines all the others. In the case of sugar cane, neighbouring plantation lines must be $3m$ distant from each other.

Because of condition (2), namely “the tractor cannot make curves that are under $50m$ in radius”, we could start with a small curve that clings to the plot boundary. If it is an arc C (of circumference) with radius $50m$, then all parallel lines will be wider.

Suppose that the plot boundary verifies a special condition explained in Subsection 2.2, namely “both ways along the plot boundary from H to L are monotonically decreasing”. Thus C should go round H , but even in this case there are infinitely many choices of an arc like that.

This contrasts with condition (4): “there must be the least possible number of plantation lines”. Hence our strategy is to trace the longest plantation line that passes through the middle of the plot under conditions (1)-(5). This will be called *the master line*.

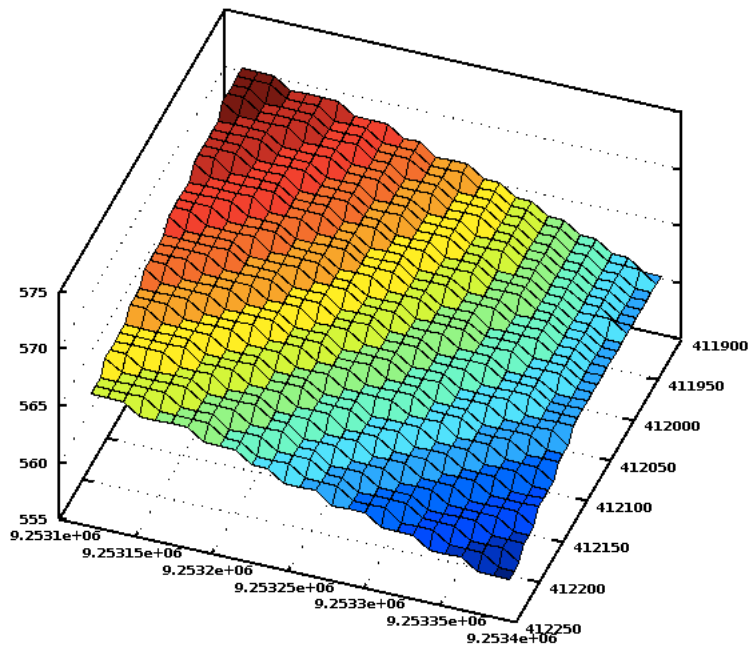


Figure 2: 3D-rendering of a level curves-file.

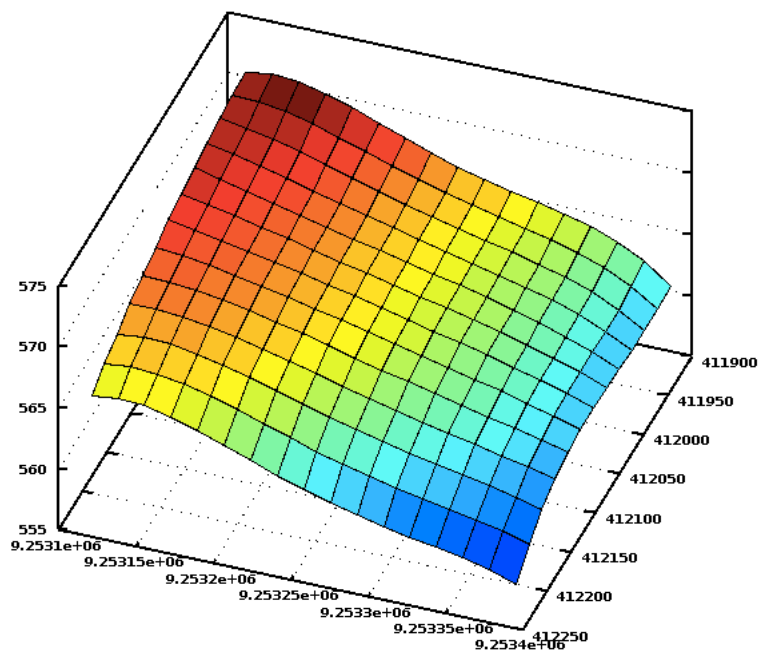


Figure 3: Plotting the polynomial $z(x,y)$ with the Matlab function `surf(x,y,z)`.

Due to condition (1), there must be little variation of the z -coordinate along the master line. In particular, condition (5) must hold. After having drawn our first choice, we can gradually perturb our master line in order to obtain the least number of parallels, as required by condition (4).

One detail that we have omitted so far: even on the most modern driverless tractors you still cannot programme a 3D-plantation line. This is just because the plot region must have a gentle slope. Moreover, condition (1) implies that the master line is almost planar. Therefore, if you programme a 2D-line on the tractor it will already follow a 3D-way by crossing the level curves.

Mathematically, the master line can be defined as a smooth regular curve $C : [0, \ell] \rightarrow \mathbb{R}^2$, where ℓ is its length. Without loss of generality, $C(t) = (x(t), y(t))$ is parametrized by its arclength. Its osculating circle has radius greater than 50 for all $t \in [0, \ell]$. This implies that the parallels $C(t) + s(-y'(t), x'(t))$ are smooth and regular for $-50 < s < 50$.

Now one must take enough parallels to cover the whole plot and guarantee that in its inside none of them will have an osculating radius under 50. It is indeed a laborious task in the case of a general curve C .

But suppose that $[0, \ell]$ admits a partition $0 < \ell_1 < \ell_2 < \dots < \ell_k = \ell$ such that the restriction of C to the sub-intervals is an arc of circumference. Two consecutive arcs must be tangent because C is smooth. We call it *arc-piecewise* (apw). In this case, a classical result of Geometry states that all parallels to C are also apw and their osculating centres will remain fixed. See Figure 4.

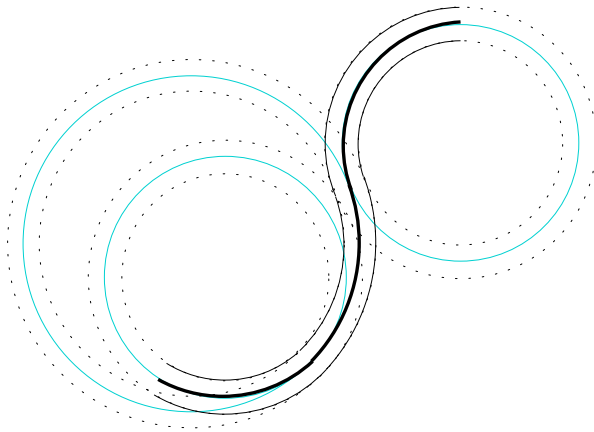


Figure 4: Parallel curves to an apw-master line.

If a curve is apw, its osculating radius is always the same of the local arc. In our programme we obtain a master line that is apw. Since its parallels have all the same osculating centres, we first guarantee that none of them is in the plot. Afterwards we take the one with minimum distance d to the plot. If $d < 50$ we gradually widen the corresponding arc of C until that centre is sufficiently moved away, namely $d \geq 50$. The process is repeated until all centres are farther than 50 from the plot.

This procedure is easy to implement when the master line consists of a single arc. We can just fix an internal point of the arc and change its extremities a bit towards alignment. Such a task will be called “compute new extremities” in Algorithm 3 (see below).

4 Computing The Master Line for a Convex Plot

As described in Section 3, the master line is the longest plantation line that passes through the middle of the plot under conditions (1)-(5). Herewith we explain our algorithm for the case of a convex plot. Concave plots will be discussed in Section 5.

The vertices of the plot are stored in the variable **pts**. They determine a polygon, which is the boundary of the plot. We need to go round the polygon and find the points that attain H and L . If there are more than two, we take the pair with the farthest detachment.

In order to inspect several points of the boundary, we go round it at a pace of at most $10m$ step. All these step-points are stored in the variable **Pts**, which contains **pts**.

The corresponding z -values on **Pts** are computed via the 2D-polynomial $z(x, y)$ and stored in **zc**. We have $H = \max(zc)$ and $L = \min(zc)$ attained at **Pts(h)** and **Pts(l)**, respectively. The “middle of the plot” is then **mp**, as in Algorithm 2. The interpolating polynomial $z(x, y)$ at **mp** gives **zp**, and then we look for another two points on the boundary that are at level **zp+de** and are the farthest from each other. We begin with **de** = 2 because in practice more than $2m$ already violates condition (1).

Once we get these three points, a prototype of the master line is ready for us to begin with. Such a prototype is never shown to the user, but it is exemplified by **plot3.txt** in Figure 5.

The level curves in Figure 5 are of the polynomial $z(x, y)$, not of the level curves-file. Our programme always shows both in separate windows for comparison.

ALGORITHM 2: Obtaining the middle point of the plot

Input: H,L,Pts,z**Output:** mp,zp

zc=z(Pts);

[L,l]=min(zc);

[H,h]=max(zc);

mp=(Pts(h)+Pts(l))/2;

zp=z(mp);

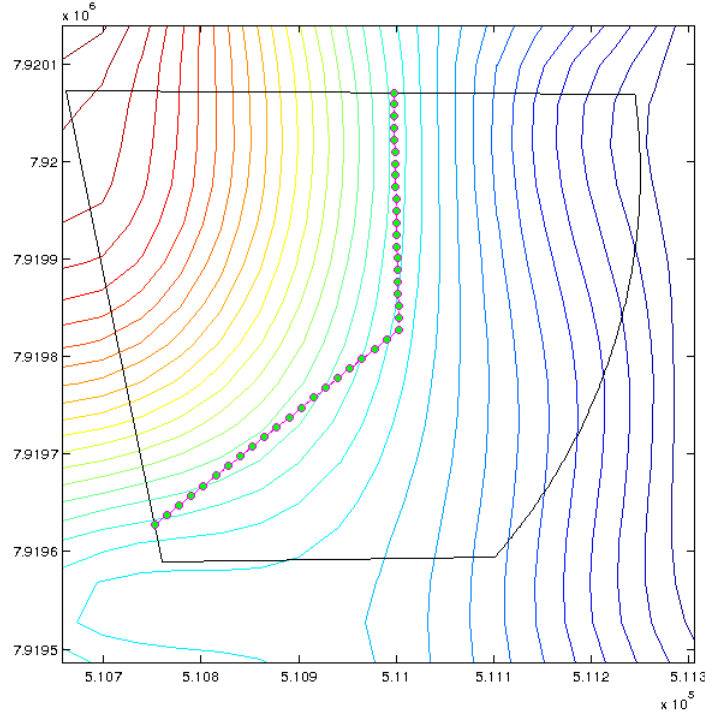


Figure 5: A prototype of the master line.

The initial choice $de = 2$ also guarantees that the three strategic points of the prototype will never be collinear (mp and the extremities). Hence, they determine an arc of circumference. This arc is then adjusted to optimize the master line according to conditions (1)-(5) as described in Algorithm 3. Of course, we use condition (3) in order to count the number of plantation lines that cover the plot. This number is stored in the variables **nbf** and **naf**, which count it before and after the while-loop, respectively.

In the next section we shall apply Algorithm 3 again. Profiting from already implemented codes is another reason why our programme was written with only 573 lines.

5 Computing The Master Line for a Concave Plot

As explained in Section 4, the plot boundary is a polygon. Its vertices are stored in the variable **pts**. From classical Geometry we know that a set is convex exactly when it coincides with its convex hull. Our sub-programme **cvxhull** computes it for **pts**. If the sets do not coincide, then the plot is concave.

It is worthwhile commenting that **cvxhull.m** has only 30 lines of source code. As a matter of fact, this is mostly due to the plots being handled in 2D. Generating the convex hull of a spatial set is far much harder a task (see [5]).

Concave plots can be subdivided into two or more convex regions. It is precisely at this moment that our programme asks the user to intervene. They have to choose strategic points that subdivide the plot into “nice” convex regions. Intuitively, “nice” means that we should get an apw-master line that verifies conditions (1)-(5) for the plot as a whole.

Namely, there are infinitely many ways of subdividing the plot into convex regions. However, instead of considering them separately, they must be viewed as parts of an entire plot. Since this is highly intuitive we do believe that unsupervised methods will fail to make good subdivisions.

In practice the user will have to select at most three pairs of points. It is done by clicking the mouse on chosen points of the polygon. This task becomes trivial with time, but the user can save the chosen points as soon as a good subdivision is found. Figure 6 exemplifies this task on **level_curves2.txt** with a very simple concave plot.

ALGORITHM 3: Optimizing the master line

Input: de, master line, Pts

Output: optimized master line

$nbf = 999$; $naf = 999$;

c = centre of master line;

$d = \min |c - Pts|$;

while ($(d < 50$ or $naf \leq nbf)$ and $de > 0$) **do**

$nbf = naf$;

$de = de/2$;

 compute new extremities;

c = centre of new master line;

$d = \min |c - Pts|$;

$naf = (\max |c - Pts| - d)/3$;

end

if $\max |angle| \leq 5^\circ$ **then**

 master line = optimized master line;

else

 printf("Angle Not OK");

end

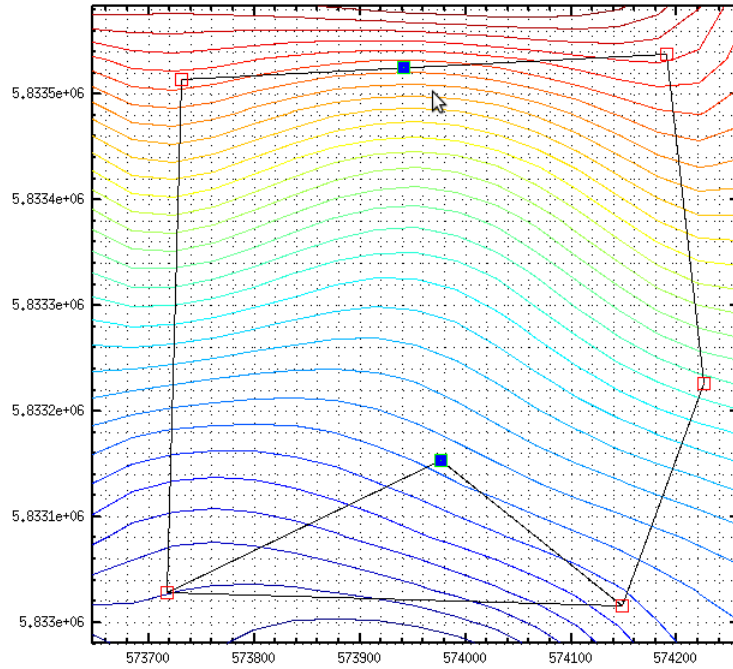


Figure 6: Choosing strategic points to subdivide the plot.

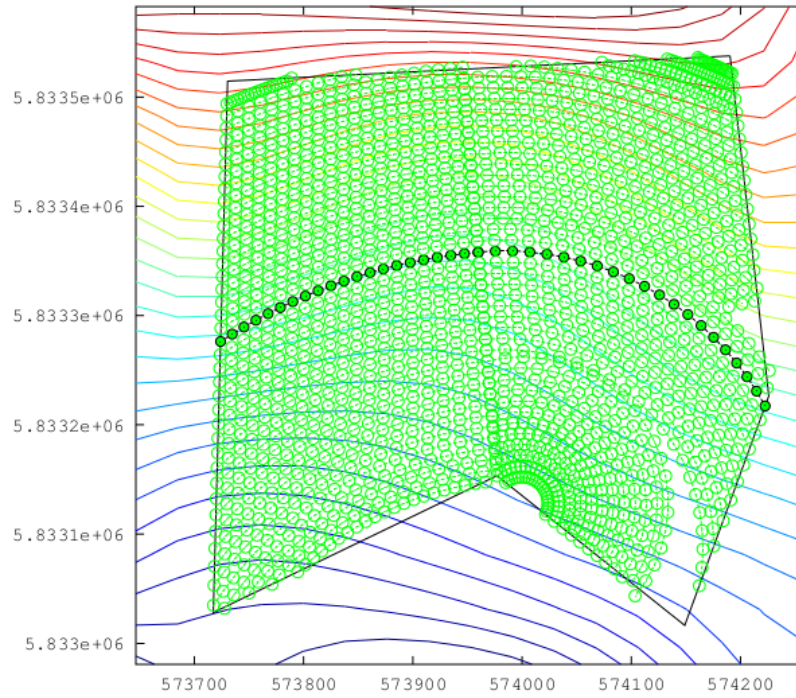


Figure 7: A bad choice of extra points from Figure 6.

The choice from Figure 6 resulted in a bad master line, as depicted in Figure 7.

That is why we have not saved the chosen points. After some tries the best choice was finally found and stored in `eplot2.txt` (contained in our zip-file).

6 Conclusions

We hope to have contributed with the semi-automation of a task that has been performed manually for many decades. Of course, our programme has limitations and `level_curves4.txt` shows an example.

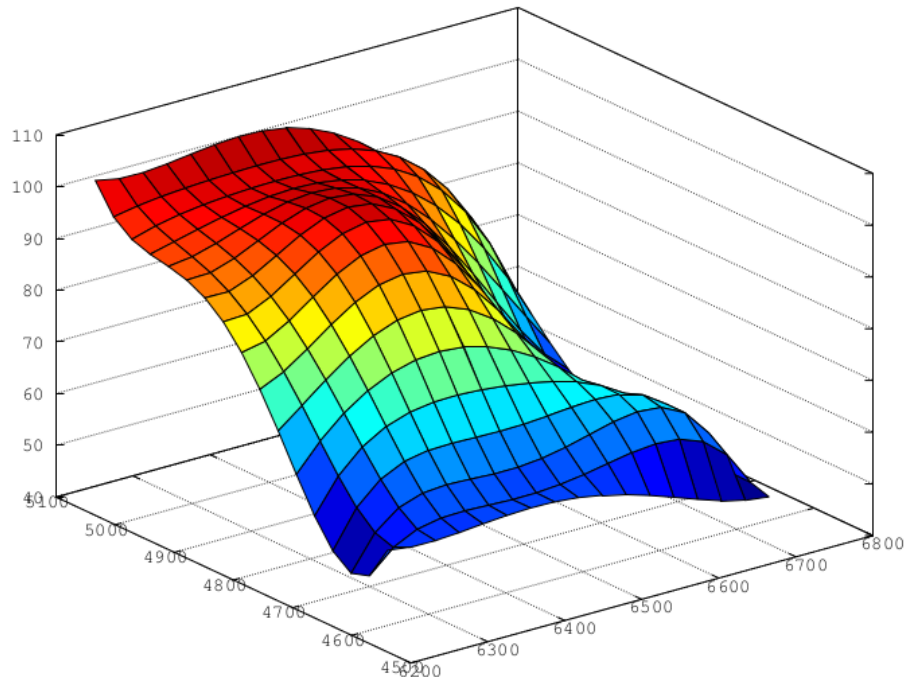


Figure 8: A slight hill inside the plot.

However, Figure 8 indicates a problem that commonly remains unsolved even if tackled manually by good experts: the inside of the plot is hilly. In practice, such a plot is subdivided in two subplots by a lane and the

tractor has to perform manoeuvres on the lane when it passes from one region to the other.

It is still unclear whether the ideal lane can be found by a computer programme. As a matter of fact, we believe that some tasks will ever remain to be accomplished manually. This is not exclusive to Agriculture. For instance, Computer Aided Diagnosis (CAD) has been an indispensable help in Medicine for many decades already. In the case of digital(ized) mammographies, CAD systems can detect and classify nodules as normal, benign or malignant, this latter being the only kind that requires extraction. The automatic classification makes use of several techniques. One of them is the Wavelet Transform (see [13]).

Anyway, no matter how good CAD systems are, they in fact serve as tool to supplementary analysis that helps radiologists achieve preciser diagnoses. The ultimate conclusions must still rely on these professionals. For details, see [10, 11, 7].

References

- [1] Michelle Cristina Araujo Picoli, Rubens Augusto Camargo Lamparelli, Edson Eyji Sano, Jefferson Rodrigo Batista de Mello, and Jansle Vieira Rocha. Effect of sugarcane-planting row directions on alos/palsar satellite images. *GIScience & Remote Sensing*, 50(3):349–357, 2013.
- [2] Tijmen Bakker, Hendrik Wouters, Kees Van Asselt, Jan Bontsema, Lie Tang, Joachim Müller, and Gerrit van Straten. A vision based row detection system for sugar beet. *Computers and Electronics in Agriculture*, 60(1):87–95, 2008.
- [3] Timothy A. Davis and Ekanathan Palamadai Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Transactions on Mathematical Software*, 37(3):36:1–36:17, sep 2010.
- [4] Carl De Boor. *Polynomial interpolation in several variables*. Springer USA, 1994.
- [5] Mingcen Gao, Thanh-Tung Cao, Ashwin Nanjappa, Tiow-Seng Tan, and Zhiyong Huang. gHull: a GPU algorithm for 3D convex hull. *ACM Transactions on Mathematical Software*, 40(1):3:1–3:19, oct 2013.
- [6] Federico Giudiceandrea, E Ursela, and Enrico Vicario. A high speed ctscanner for the sawmill industry. In *17th International Nondestructive Testing and Evaluation of Wood Symposium Sopron*, pages 14–16, Sopron Hungary, 2011.
- [7] Ricardo Souza Jacomini, Marcelo Zanchetta Nascimento, Rogério Daniel Dantas, and Rodrigo Pereira Ramos. Comparison of pca and anova for information selection of cc and mlo views in classification of mammograms. In Hujun Yin, José A. F. Costa, and Guilherme Barreto, editors, *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, volume 7435 of *Lecture Notes in Computer Science*, pages 117–126. Springer Berlin Heidelberg, 2012.
- [8] Robert G. Keys. Cubic convolution interpolation for digital image processing. *Computers and Electronics in Agriculture*, ASSP-29(6):1153–1160, 1981.
- [9] Daniel S. Maynard, Mark J. Ducey, Russell G. Congalton, John Kershaw, and Joel Hartter. Vertical point sampling with a digital camera: Slope correction and field evaluation. *Computers and Electronics in Agriculture*, 100:131–138, 2014.
- [10] M. Z. Nascimento, A. S. Martins, L. A. Neves, R. P. Ramos, E. L. Flores, and G. A. Carrijo. Classification of masses in mammographic image using wavelet domain features and polynomial classifier. *Expert Systems with Applications*, 40(15):6213–6221, 2013.
- [11] Rodrigo Pereira Ramos, Marcelo Zanchetta Nascimento, and Danilo Cesar Pereira. Texture extraction: An evaluation of ridgelet, wavelet and co-occurrence based methods applied to mammograms. *Expert Systems with Applications*, 39(12):11036–11047, sep 2012.
- [12] Kanae Tanigaki, Tateshi Fujiura, Akira Akase, and Junichi Imagawa. Cherry-harvesting robot. *Computers and Electronics in Agriculture*, 63(1):65–72, 2008.
- [13] Carl Taswell and Kevin C. McGill. Algorithm 735: Wavelet transform algorithms for finite-duration discrete-time signals. *ACM Transactions on Mathematical Software*, 20(3):398–412, sep 1994.